We continue our exploration of database design in part two of this 14-part series, Process-Driven Data Design.

DATA FLOW DYNAMICS

by Frank Sweet

Data flow through files like water in an irrigation system. Some pipes carry much volume, others less. In parts of a database, records race along in a rush while in other places they pool into reservoirs with little perceptible movement. Yet everywhere they follow the rule,

POOLTIME × FLOWRATE = POPULATION.

In other words, if we mentally isolate a section of a system, we'll find that records follow a simple, steady-state law: the length of time that records spend in any section, multiplied by the rate at which records flow into the section, is numerically equal to the number of records currently in the section.

Database designers must know how to derive and use the consequences of this law. It can be applied in all steps of our job, from initial user interviews to physical file design. In a moment, we'll illustrate this with a sample problem, but first let's examine what we mean by steady state.

Steady state means that the flowrate of records entering a section is equal, over the long pull, to the rate at which they leave. Say we have a file where 1,000 new records are added each month. The file is in a steady state if, over a long period (a year, for instance), an average of 1,000 records is also removed from the file each month. Some records might be removed soon after they're added, while others could stay on file forever. Nevertheless, if 12,000 are added each year and approximately 12,000 removed, the file fits our definition. With few exceptions, every system we'll study is in a steady state. Look at it this way: if more records were removed each month than were added, the file would soon disappear entirely, thus reaching a steady state with a population of zero. If the reverse were true, it would quickly reach the capacity limit of the medium on which it's stored.

Another aspect of steady state is that the section we study contains no unidentified data sinks or data sources. Consider a credit-checking procedure where credit applications are funneled into a department for review. If we know that their total backlog is more or less constant from year to year, we conclude that, on average, just as many emerge each month as are sent in. The only way it could be otherwise would be if somewhere in the department applications were being destroyed, never to be seen again (data sink), or if someone in the department were producing new applications internally (data source). Both are unlikely.

If the number of records-out always equals the number of records-in, why did the formula use flowrate into a section, rather than out of it? Either is valid, but records-in (new records added) are usually easier to measure. A typical vendor file, say, receives new records whenever the firm first deals with a new supplier. Inactive vendors are commonly purged once a year. If we're researching the system, it's often easier to estimate how many new vendors are added each month than to find the purge rate. And unless the firm is withering away or in a state of uncontrolled growth, we can be confident that, over the long haul, the number of inactive records purged each year will be close to the number of new records added.

VALUE OF THE FORMULA

The importance of the formula is that given any two of the terms, we can easily compute the third. Given all three, we can cross-check them against each other. But enough lecturing. You now know enough to solve the following problem. It was taken from actual experience and, though the application is disguised, the numbers are authentic. It can be solved by using the flowrate formula as an instrument for applying common sense. In part three of this series, I'll present the answer I actually encountered.

The Case of the Unreserved Work Orders. As part of a materials management project (MRP, shop loading, etc.), we were designing a work order reservation system. New manufacturing jobs, or work orders, had their component requirements checked against available inventory before being issued to the shop. The idea was to avoid starting a job for which parts were missing. Instead, the job would be held while needed parts were expedited. The manual system was working well and our task was simply to automate it by checking if each order's parts were in stock, releasing the job to the shop if they were, otherwise printing an expedite list.

It was obvious we'd need a file of pending work orders—those held awaiting arrival of needed parts—and to avoid the order-number wraparound we needed to know just how long, on average, we could expect pending orders to stay on file. Investigating the manual system, we found 200 work orders scattered around in expediting, awaiting parts arrivals. Although individual orders came and went, the total pending backlog had been constant at about 200 for as long as users could recall. Further, we were told that about 40 new work orders were issued to the floor each month and that, on average, each waited for about 10 days for parts to arrive before being released.

How long will orders wait on file, on average, before expediting scares up the parts for each one?

In part three we'll offer the answer we found and explore the world of "Objects and Events." ◉

Frank Sweet is corporate manager of data administration for the Charter Co., Jacksonville, Fla.